15-451/651 Algorithms, Fall 2017

Homework #5 **Due: November 5, 2017**

(25 pts) 1. **Shortest Cycle**

You're given a directed weighted graph $G = (V, E, w)$. There are no self-loops or multi-edges. Here $w_{i,j} \geq 0$ is the length of edge $(i, j) \in E$. For $(i, j) \notin E$ we have $w_{i,j} = \infty$.

Your goal is to find the length of the shortest cycle in G.

(a) Give an algorithm that runs in $O(nD(n, m))$, where $D(n, m)$ is the running time of Dijkstra's algorithm on a graph of n vertices and m edges.

Solution: Given a vertex v here's how we can find the length of the shortest cycle through v (if there is one) in $O(D(n, m))$ time.

Run Dijkstra's algorithm with v as the starting vertex. Label each vertex u with t_u, the computed distance from v. Now construct a new graph $G^R = (V, E', w')$ as follows. $E' = \{(v, u) \mid (u, v) \in E\}$, and $w'_{v,u} = w_{u,v}$ if $(v, u) \in E'$. Now run the same algorithm starting from v in G^R. This time for vertex u call the result f_u which is the distance from v to u in G^R which is the same as the distance from u to v in G.

Consider the quantity $X_v = \min_u f_u + t_u$. This is the length of the shortest cycle that runs through vertex v. Here's the proof. Say the shortest cycle that runs through v contains a vertex x. (It must contain some vertex other than v because there are no self-loops in G.) Then this cycle's length will be found when considering $f_x + t_x$, which is the length of the cycle. ■

Note $m \leq D(n, m)$, so the time to construct G^R is within the $O(D(n, m))$ bound.

Now to solve the original problem, run the above algorithm for all possible starting vertices v. This runs in the required time bound and finds the shortest cycle.

(b) Another solution to this problem is obtained by tweeking the Floyd-Warshall algorithm. Initialize $A^0_{i,j} = w_{i,j}$. Note that the diagonal elements are all initially ∞ (unlike the standard version of Floyd-Warshall).

```
For k = 1, 2, ... n do:
    For i = 1, 2, ... n do:
        For j = 1, 2, ... n do:
            A^k_{i,j} ← min(A^{k-1}_{i,j}, A^{k-1}_{i,k} + A^{k-1}_{k,j})
```

Prove that after running this $O(n^3)$ algorithm the minimum length cycle is $\min_{1 \leq i \leq n} A^n_{i,i}$.

Solution: The proof that this algorithm works is analogous to the proof presented in lecture that the Floyd-Warshall algorithm works. We will prove the following statement by induction:

Theorem: $A^k_{i,j}$ is the shortest path from i to j using only intermediate vertices in the set $\{1, 2, \ldots, k\}$. This holds even if $i = j$.

Proof: The proof is by induction on k. The base case, $k = 0$, is true by the definition of $A^0_{i,j}$. Recall the notation that $[1, k] = \{1, 2, \ldots, k\}$.

Now let's assume the theorem is true for $k-1$, and prove it for k. The shortest path from i to j using intermediate vertices in $[1, k]$ either uses k or it does not use k. If it uses k how long is it? (If it does use k it can use it exactly once, WLOG. A cycle from k back to k has non-negative length and can be removed without making the path longer.) So this path has to get from i to k using only vertices in $[1, k-1]$. Then it has to get from k to j using only vertices in $[1, k-1]$. Therefore its length can be computed as $A^{k-1}_{i,k} + A^{k-1}_{k,j}$.

On the other hand, if the shortest path from i to j does not use k as intermediate vertex, then its shortest path length is $A^{k-1}_{i,j}$.

Thus, the assignment $A^k_{i,j} \leftarrow \min(A^{k-1}_{i,j}, A^{k-1}_{i,k} + A^{k-1}_{k,j})$ computes $A^k_{i,j}$ correctly.

Note that all of this this reasoning applies when $i = j$ ■

It follows from the theorem that when the algorithm terminates the shortest cycle through a vertex i is $A^k_{i,i}$.

(25 pts) 2. **Triangle Game**

Let $T = (a, b, c)$ be a triangle in the plane where $a = (-1, 0)$, $b = (1, 0)$ and $c = (0, 2)$. Player I (the "evader") and Player II (the "searcher") play the following zero-sum game. The evader picks a point X in T and simultaneously the searcher picks a point Y in T. The payoff to the evader is $\|X - Y\|^2$, the square of the Euclidean distance between X and Y.

Find the value v of this game (from the evader's viewpoint). Prove that it's correct by giving a strategy for the evader that achieves at least v in expectation, and one for the searcher that prevents the evader from earning more than v in expectation.

Solution: In this game, for both players, a pure strategy is to pick one point, and a mixed strategy is defined by a probability distribution over all the points of the triangle.

We'll use the approach alluded to right before section 2.3.1 in the notes for lecture 14 on zero sum games. A lower bound LB on the value of the game can be obtained as follows. The maximizing player (the evader in this case) picks some mixed strategy. Given this mixed strategy, the minimizing player picks the *pure* strategy that minimizes the payoff. The result is a lower bound on the payoff for the maximizing player.

Conversely, an upper bound UB can be obtained by specifying some mixed strategy for the minimizing player, then given this, the maximizing player picks a pure strategy that maximizes the payoff.

So the approach to solving this problem is to obtain a lower and an upper bound that are equal, which gives the value of the game.

Let's start with the upper bound, because that is simpler. An obvious place to start is for the searcher to pick the point in the triangle that is equidistant from the three corners. This is the

point $(0, 3/4)$ which is $5/4$ away from each of the three corners. After this choice, the evader can do no better than going to any one of the three corners, and obtain a payoff of $25/16$. So we have found an upper bound of $25/16$.

For the lower bound it's natural to consider distributions that only go to the three corners. Also, by symmetry, let's limit our search to strategies that give $(-1, 0)$ and $(1, 0)$ the same probability, p, and $(0, 2)$ probability $1 - 2p$. With this strategy, what pure strategy will the searcher pick? And what will the payoff be? Let the point be (x, y). What's the expected payoff for the evader in this case?

$$f(p, x, y) = p(1 - x)^2 + p(1 + x)^2 + 2py^2 + (1 - 2p)(2 - y)^2.$$

Note that $(1 - x)^2 + (1 + x)^2 = 2 + 2x^2$ is minimized when $x = 0$. What about y? What value of y minimizes this?

$$f(p, y) = 2p + 2py^2 + (1 - 2p)(2 - y)^2 = 2p(1 + y^2) + (1 - 2p)(2 - y)^2$$

So given p we want to find the y that minimizes this. (This is what the searcher will do.) Simplifying we get:

$$f(p, y) = -6p + 4 + y^2 - 4y + 8py$$

Now taking the derivative w.r.t. y, we get

$$\frac{df}{dy} = 2y - 4 + 8p$$

Setting this to 0 we get $2y = 4 - 8p$, or $y = 2 - 4p$. Let's try a p such that the resulting y value is $3/4$. That is $p = 5/16$.

$$f(5/16, 3/4) = -6(5/16) + 4 + (3/4)^2 - 4(3/4) + 8(5/16)(3/4) = 25/16$$

So with this mixed strategy: with probability $5/16$ go to (-1,0) and with probability $5/16$ go to (1,0) and with probability $6/16$ go to $(0, 2)$ the evader obtains an expected value of 25/16. This is a lower bound on the game. Since it's the same as the upper bound we proved earlier, this is the value of the game.

(25 pts) 3. **Ice Cream**

The great nation of Glacia has hired you as a contractor to figure out how to maximize the national production of ice cream, the country's primary export. The country has a set of cities S of size $|S| = n$, where each city has either some cream-producing farms, or some factories to produce and package the ice cream, or neither. In particular, city i can produce r_i units of cream and can process f_i units of ice cream in its factories, where either $r_i = 0$ or $f_i = 0$ (or both) for every city.

However, due to local agreements with various shipping companies, it isn't always possible to move resources from every city to every other city. In particular, there are m shipping companies, and shipping company j has agreements with a set of cities $c_j \subseteq S$ and may ship freely into and out of any city in this set, but not to any other cities. So that the cream doesn't

go bad, at most two shipping companies may handle a given unit of cream to transport it from its source farm to its destination city. The goal is to decide which shipping companies should move cream between which cities to maximize the national production of ice cream.

For example, suppose there are four cities, 1, 2, 3, 4, and 5, with the following cream-production and cream-processing capacities:

City	r_i	f_i
1	6	0
2	2	0
3	0	0
4	0	5
5	0	3

There are three shipping companies, A, B, and C, with the following sets of agreements:

Shipping Company	c_j
A	$\{1, 3, 4\}$
B	$\{2, 3, 4\}$
C	$\{2, 5\}$

In this case, an optimal arrangement provides a total production of 7 units of ice cream, using A to ship 5 units of cream from 1 to 4 and using C to ship 2 units of cream from 2 to 5. Note that we're not allowed to have A ship 1 unit of cream from 1 to 3 that B transports to 2 and C transports to 5; however, if C had an agreement with city 3, then we could augment the above arrangement by having A transport 1 unit from 1 to 3 which C then transports to 5 a total production of 8.

This problem lends itself well to a network flow solution; however, it turns out that there are actually two good constructions, and which is better is dependent on the magnitude of the number of cities n as compared to the number of shipping companies m!

(a) Provide a network flow construction (that is, describe how to construct the network graph) that works well when $m \gg n$ (e.g., $m > n^2$). In terms of big-O, how many nodes and edges does this construction have?

(b) Provide a *different* network flow construction that works well when $m \ll n$ (e.g., $n > m^2$). In terms of big-O, how many nodes and edges does this construction have?

Hint. When $n = m$, the two constructions have an asymptotically-equal number of nodes and edges.

Solution:

Part a. Create three layers A, B, and C, which each contain a node for each city i. Also create a source s and a sink t. For each city i, construct an edge $s \to A[i]$ with capacity r_i, and an edge $C[i] \to t$ with capacity f_i. For each pair of cities (i, j), if there exists a shipping company k such that $i \in c_k$ and $j \in c_k$, then construct an edge $A[i] \to B[j]$ and an edge $B[i] \to C[j]$, each with infinite capacity. (Note that i and j need not be unique.)

The three layers represent the cities in which a unit of cream can be located after a given number of transportation steps (0 steps at layer A, 1 step at layer B, and 2 steps at layer C). The edge construction reflects this: an edge $A[i] \to B[j]$ or $B[i] \to C[j]$ is only constructed if it is possible to get from city i to city j using only one shipping company. The $s \to A[i]$ edges limit the amount of cream at each city after 0 transportation steps (i.e. the amount that starts there, or the amount that the city produces) and the $C[i] \to t$ edges limit the amount of cream that ends up at each city after 2 transportations steps (i.e. the amount that is processed by that city).

In a given flow through this graph, a path $s \to A[a] \to B[b] \to C[c] \to t$ followed by a unit of cream represents the unit of cream being produced at city a, being moved by some shipping company to city b, then being moved by some shipping company to city c, where it is then processed. Note that the flow in the resulting graph is not enough information to determine which transportation companies should transport the cream between any given pair of cities; however, we can obtain this information if we annotate the $A[i] \to B[j]$ and the $B[i] \to C[j]$ edges with their associated shipping companies when we are constructing them.

This construction has $O(n)$ nodes and $O(n^2)$ edges.

Part b. Create five layers A_1, A_2, A_3, B_1, and B_2. Layers A_1, A_2, and A_3 each contain one node for each city; layers B_1 and B_2 each contain one node for each shipping company. Also create a source s and a sink t. For each city i, construct an edge $s \to A_1[i]$ with capacity r_i, and an edge $A_3[i] \to t$ with capacity f_i. For each shipping company j and city $i \in c_j$, construct four edges of infinite capacity: $A_1[i] \to B_1[j]$, $B_1[j] \to A_2[i]$, $A_2[i] \to B_2[j]$, and $B_2[j] \to A_3[i]$.

The three A layers represent the cities in which a unit of cream can be located after a given number of transportation steps (0, 1, and 2 respectively at A_1, A_2, and A_3). The two B layers represent the shipping companies that can be in posession of a given unit of cream during a given transportation step (the first step for B_1 and the second step for B_2). Again, the edge constructions reflect this: we construct an edge $A_k[i] \to B_k[j]$ only if shipping company j can pick up a unit of cream from city i, and we construct $B_k[j] \to A_{k+1}[i]$ only if shipping company j can drop off a unit of cream at city i. As before, the $s \to A_1[i]$ edges limit cream production at city i, and the $A_3[i] \to t$ edges limit cream processing at city i.

In a given flow in this graph, a path $s \to A_1[a] \to B_1[b] \to A_2[c] \to B_2[d] \to A_3[e] \to t$ followed by a unit of cream represents the unit of cream being produced at city a, then being moved by shipping company b to city c, then being moved by shipping company d to city e, where it is then processed.

This construction has $O(n + m)$ nodes and $O(nm)$ edges.

(25 pts) 4. CATS

In this programming problem you're given a rectangular board of letters. The word "cats" appears in the board if there is a sequence of four squares such that "c" is in the first one, "a" is in the second one, and so on. Each letter is next to (either horizontally or vertically) the previous one.

The *score* of a board is the maximum number of times you can find "cats" in it. That is, you find the word "cats" as described above. Then cross off those cells (they can't be used again). Then repeat this step as many times as possible. (See the examples below.) The time limit is 10 seconds.

Input:

The first line contains two space-separated integers: r and c. The next r lines are strings of c lower case letters followed by a newline. $1 \leq r, 1 \leq c$, and $r \cdot c \leq 10000$.

Output:

Print the maximum number of disjoint cats that can be simultaneously found on the board (the score described above).

Here are some sample input and output pairs:

```
4 4                                                    1 cat
cats
axxx
txxx
sxxx

1 10                                                   2 cats
stacatstac

4 4                                                    1 cat
saos
tacc
tcoa
otcs

2 10                                                   3 cats
catscatsca
atscatscat

1 1                                                    0 cats
x
```